Publications Internes de l'IRISA
ISSN : 2102-6327
PI 1980 – juillet 2011

Leader Election: From Higham-Przytycka's Algorithm to a Gracefully Degrading Algorithm

Itziar Arrieta[*] , Federico Fariña[**] , José Ramón G. de Mendívil[***] , Michel Raynal[****]
itziar.arrieta@unavarra.es raynal@irisa.fr fitxi@unavarra.es mendivil@unavarra.es

Abstract: The leader election problem consists in selecting a process (called leader) in a group of processes. Several leader election algorithms have been proposed in the past for ring networks, tree networks, fully connected networks or regular networks (such as tori and hypercubes). As far as ring networks are concerned, it has been shown that the number of messages that processes have to exchange to elect a leader is $\Omega(n \log n)$. The algorithm proposed by Higham and Przytycka is the best leader algorithm known so far for ring networks in terms of message complexity, which is $1.271\ n \log n + O(n)$. This algorithm uses round numbers and assumes that all processes start with the same round number. More precisely, when round numbers are not initially equal, the algorithm has runs that do not terminate.

This paper presents an algorithm, based on Higham-Przytycka's technique, which allows processes to start with different round numbers. This extension is motivated by fault-tolerance with respect to initial values. While the algorithm always terminates, its message complexity is optimal, i.e., $O(n \log n)$, when the processes start with the same round number and increases up to $O(n^2)$ when all processes start with different round number values. We call *graceful degradation* this additional property that combines fault-tolerance (with respect to initial values) and efficiency.

Key-words: Asynchronous system, Deterministic algorithm, Distributed algorithm, Fault-tolerance, Initial value fault, Leader Election, Message complexity, Round number.

Un algorithme d'élection d'un leader

Résumé : *Ce rapport présente un algorithme d'élection sur un anneau.*

Mots clés : *Anneau, Communication asynchrone, Election, Fautes sur les valeurs initiales.*

---

[*] Universidad Pública de Navarra, Pamplona, Spain
[**] Universidad Pública de Navarra, Pamplona, Spain
[***] Universidad Pública de Navarra, Pamplona, Spain
[****] Institut Universitiare de France, Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes

1 Introduction

The leader election problem In the *leader election* problem the processes that compose a message-passing distributed system have to choose one of them to be their leader. The leader can be any of them. The fundamental property is that a single leader is eventually elected. In addition to mutual exclusion [11], leader election is one of the most important problems of the class of *symmetry-breaking* problems.

It is important to observe that symmetry cannot be broken "from nothing", i.e., in systems in which processes are "identical". As an example, if all processes have the same number of neighbors in the communication graph and have no name (the system is then anonymous), there is no way to break the symmetry among processes and therefore the leader election problem is impossible to solve [1, 2, 12]. Hence, we consider here that each process has an identifier and no two processes have the same identifier. Moreover, identifiers have to be usable, namely, they are totally ordered and can consequently be compared.

Election on rings: a short historical perspective The leader election problem has been introduced by Le Lann in [9], which also presents a leader election algorithm for bidirectional rings whose message complexity is $O(n^2)$. This result has been improved for the average case by Chang and Roberts [5], who designed an algorithm whose average case message complexity is $O(n \log n)$ and worst case message complexity is $O(n^2)$. Then, Hirschberg and Sinclair designed an optimal algorithm for bidirectional rings, i.e., an algorithm whose message complexity is always $O(n \log n)$.

While Hirschberg and Sinclair conjectured that $O(n^2)$ was a lower bound on the number of messages for election in unidirectional rings, simultaneously, independently and using distinct approaches, Peterson on the one hand [10] and Dolev, Klawe and Rodeh on the other hand [6], disproved this conjecture by presenting election algorithms for unidirectional rings requiring at most $2n \log n + O(n)$ messages. Then, Dolev Klawe and Rodeh improved their algorithm to obtain a $1.5\ n \log n + O(n)$ message complexity while Peterson improved his algorithm to obtain $1.441\ n \log n + O(n)$ message complexity. Applying their technique to Peterson's algorithm, Dolev Klawe and Rodeh obtained an algorithm with message complexity $1.356\ n \log n + O(n)$.

Finally, using a totally new approach, Higham and Przytycka designed an election algorithm for unidirectional rings whose message complexity is $1.271\ n \log n + O(n)$ [7]. It is shown in [4] that the message complexity of the leader election problem on a unidirectional ring is $\Omega(n \log n)$. Hence, Higham-Przytycka's algorithm is optimal with respect to that complexity measure. Moreover, due to the constant 1.271, it is the best algorithm known so far when considering that criterion. However, it is not yet known which is the smallest constant c such that election can be solved on unidirectional rings with message complexity $c\ n \log n + O(n)$ (it is only known that $c \geq 0.69$ [4]).

The reader interested in election algorithms in rings, tree networks, fully connected networks or regular networks (such as tori and hypercubes) will find an in-depth investigation in Chapter 2 of Nicola Santoro's book [13] (pages 99-224).

Content of the paper Higham and Przytycka's algorithm is based on round numbers and assumes that all processes start with the same round number value. More specifically, when processes start with distinct round number values, the algorithm has runs that do not terminate.

This paper considers the possibility of initial value faults, i.e., the case where the initial values of the round numbers are faulty in the sense that not all processes start with the same round number value. We are interested in a *gracefully degrading* election algorithm for unidirectional rings. In the context of initial value faults for the election problem, graceful degradation with respect to round numbers means the following.

- The algorithm has to elect a leader (safety property) and terminate (liveness property) whatever the initial value of the round number at each process, and
- Its message complexity
  - has to be as good as the best known algorithms (i.e., $O(n \log n)$) when all processes start with the same round number value ("good" initial values), but
  - is allowed to increase up to $O(n^2)$ when processes start with different round number values ("bad" initial values).

This paper presents a gracefully degrading election algorithm. Let us observe that, in our context, graceful degradation means that the processes are allowed to use more messages only when their initial round number values are faulty in the sense they are not equal. Hence, graceful degradation captures an algorithmically interesting tradeoff between the initialization of local variables and the cost of an execution measured by the number of exchanged messages.

Roadmap The paper is made up of five sections. Section 2 introduces the system model. Section 3 presents the gracefully degrading algorithm, while Section 4 proves its properties. Finally, Section 5 presents some concluding remarks.

```
init: if (status_i = candidate) then send (r_i, id_i) end if.

repeat
(01)   receive (r, id);
(02)   if (status_i = candidate)
(03)      then case
(04)              r = r_i: case
(05)                          id_i(-1)^{r_i} > id(-1)^r: r_i ← r_i + 1; send (r_i, id_i)
(06)                          id_i(-1)^{r_i} < id(-1)^r: status_i ← relaying
(07)                          id_i(-1)^{r_i} = id(-1)^r: status_i ← leader
(08)                       end case
(09)              r > r_i: status_i ← relaying; send (r, id)
(10)              r < r_i: discard the message
(11)           end case
(12)      else if (status_i = relaying) then send (r, id) end if
(13)   end if
until (status_i = leader) end repeat.
```

Figure 1: Gracefully degrading leader election algorithm (code for p_i)

2 Computing Model

a) Process model: The system is made up of n asynchronous sequential processes denoted p_0, p_1, ..., p_{n-1}. The integer i associated with p_i is its index. Indexes are used only from an external observer point of view in order to make the presentation simpler. Indexes are not known by processes. "Asynchronous" means that each process proceeds at its own speed, which can be arbitrary. Moreover, no process has a priori information on the way other processes progress.

A process p_i has an identifier id_i (id_i is a constant). No two processes have the same identifier and the identifiers are totally ordered. A process initially knows only its own identifier and the fact that identifiers can be compared. Initially, a process knows neither the identifier of the other processes nor the total number of processes n.

b) Communication model: The processes are connected by a unidirectional reliable asynchronous ring on which they can send and receive messages. "Asynchronous" means that the transfer delay of each message is arbitrary, while "reliable" means that all transfer delays are finite (no message loss) and that messages are neither corrupted nor duplicated.

"Unidirectional ring" means that messages can flow in only one direction. More precisely, a process can receive messages only from its "left" neighbor on the ring and can send messages only to its "right" neighbor on the ring. Hence, the ring has a single sense of direction (the notion of "left/right" is the same for all processes [13]). Hence, without ambiguity, the sender or the destination process of a message are left implicit when a process invokes operations send() or receive().

Differently from the model used in other papers (e.g., [6, 7, 9], the channels are not required to be FIFO (First In First Out).

3 The Algorithm

The proposed leader election algorithm is described in Figure 1. As explained below, one of its underlying principles is a priority rule based on the parity of the current round number. This priority rule has been introduced by Higham and Przytycka in [7].

3.1 Description of the algorithm: Local variables

In addition to its constant identifier id_i, each process p_i manages two local variables: $status_i$ and r_i.

- $status_i$ denotes p_i's current status. Namely it is a value in the set $\{candidate, relaying, leader\}$. If a process p_i wants to compete to become the leader its initial value is $candidate$, otherwise it is $relaying$. It is assumed that at least one process initially competes to be the leader.

  At the end of an execution, a single process p_ℓ (from the candidate processes) will be such that $status_\ell = leader$. We assume that, when a process is elected as a leader, it sends a message on the ring to inform the other processes that it is the leader so that they locally stop their participation in the election algorithm.

- r_i denotes the current round number of p_i. It can be initialized to any integer.

3.2 Description of the algorithm: Behavior of a process

The aim of the algorithm is to elect a candidate process whose identifier is an extremum among the current candidates. To that end each process p_i does the following.

If p_i is candidate, it sends a message containing the pair (r_i, id_i) to its right neighbor. Moreover, if it has not yet executed the **init** statement when it receives a message carrying a pair (r, id), p_i executes this statement before processing the message it has just received.

When a process p_i receives a message (r, id) (line 01), it forwards it to its right neighbor if $status_i = relaying$ (line 12). In this case, p_i was not candidate or has been eliminated from the competition before receiving that message.

Otherwise, if $status_i = candidate$ (line 02), there are three cases to consider.

- $r = r_i$ (line 04). There are three sub-cases.
  - If the round number is even and $id_i > id$ or the round number is odd and $id_i < id$ (predicate $id_i(-1)^{r_i} > id(-1)^r$, line 05), p_i discards message (r, id), increments its round number r_i in one unit and sends a new message (r_i, id_i) to its right neighbor. This message indicates that p_i is still a candidate and is trying to eliminate another candidate. If p_i later receives its own message from its left neighbor (hence this message will have passed through all the other processes) it will consider itself as the leader.
  - If the round number is even and $id_i < id$ or the round number is odd and $id_i > id$ (predicate $id_i(-1)^{r_i} < id(-1)^r$, line 06), p_i considers that id_i has a smaller priority than id. Consequently, it sets $status_i$ to the value $relaying$. Note that message (r, id) is not forwarded. If r is currently the highest round number, there will be one message (r_i, id_i) (which has been previously sent by p_i) that will cause another process p_k (such that $id_k = id$ or any other process with $id_i(-1)^{r_i} < id_k(-1)^{r_k}$ and $r_k = r$) to increment r_k upon receiving (r_i, id_i). Since r is not the highest round number (either due to the previous case or because there already existed a message with a higher round number), (r, id) is no longer useful. Consequently, forwarding (r, id) would unnecessarily increment the number of messages involved in the algorithm execution.

    The strategy used in both previous items, where the priority is defined from the parity of the current round number and the values of id_i and id (the process identifier received in the message) has first been proposed by Higham and Przytycka in [7]. In that sense, the proposed algorithm is somehow inspired by their leader election algorithm.
  - If the message received by p_i is such that $(r, id) = (r_i, id_i)$, then this message was generated by p_i and has passed through all the processes of the ring. Process p_i considers itself as the leader (line 07). As mentioned before, p_i can then send a message along the ring so that the rest of processes learn which process has been elected and locally terminate their execution of the election algorithm.
- $r > r_i$ (line 09). Due to its higher round number, the received message has priority over the last message sent by p_i. Consequently, p_i changes to status $relaying$ (thus being eliminated from the competition) and forwards the received message.
- $r < r_i$ (line 10). In that case, the received message is too old, and p_i discards it.

It is worth noting that the execution of different actions depending on whether $r > r_i$ or $r < r_i$ (forwarding the message in the former case and discarding it in the latter case) is what allows processes to start with different round numbers. In contrast, in the solution of Higham and Przytycka [7] round numbers are compared only to check whether they are equal or different. More specifically, in the basic algorithm presented in [7], when a process receives a message whose round number does not match with that of the last sent message, it simply forwards it. As a result, if all processes start with a different round number the algorithm will never terminate, since messages will be perpetually forwarded around the ring. Similarly, the improved version described in [7], which uses early promotion by distance in odd rounds and early promotion by witness in even rounds, will also suffer from non-termination if all the initial round numbers are all different from each other and even, because messages are neither discarded nor promoted to the next round.

4 Proof of the Algorithm

This section proves first that the algorithm is correct, i.e., there cannot be more than one process elected as leader (Theorem 1) and the algorithm always terminates its execution electing a leader (Theorem 2). Then, we show that the number of messages is upper bounded by (a) $O(n \log n)$ when the processes start with the same round number (Theorem 3) and (b) $O(n^2)$ when the processes start with arbitrary round numbers (Theorem 4).

Let $I = \{0, 1, \ldots, n-1\}$ be the set of process indexes. The binary operators $+$ or $-$ over I are assumed to be modulus n, i.e., for every $i \in I$, $(i \pm 1) \equiv (i \pm 1) \bmod n$. Given $i, j \in I$, $D(i, j)$ is defined as the set $\{(i+1), (i+2), \ldots, (j-1)\}$, i.e., it represents the set of indexes of processes in the ring at the right side of p_i before reaching p_j. Moreover, $D(i, i)$ is well defined: $D(i, i) = I - \{i\}$.

The set of process identifiers is denoted by $ID = \{id_i : i \in I\}$. Round numbers are assumed to belong to the set of natural numbers including 0, namely $\mathbb{N}$. For the set $\mathbb{N} \times ID$, we define the total order relation $\succ$ as follows: $(r', id') \succ (r, id) \Leftrightarrow (r' > r) \vee (r' =$

$r \wedge id'(-1)^{r'} > id(-1)^{r}$). In addition, $(r', id') = (r, id)$ if and only if $r' = r$ and $id' = id$. The total order allows us to compare messages of the algorithm, since every message m belongs to $\mathbb{N} \times ID$. Furthermore, received messages can be compared with the round number and identifier (r_i, id_i) of the receiving process p_i.

The computation state of the distributed algorithm at any time t is a record s_t (where t can be omitted when it is clear in the context) comprising the values of the variables of each process p_i in the system (where $s_t.var_i$ represents the value of the variable with name var at process p_i at time t), as well as the messages in the communication channels $s_t.c[i]$ for every $i \in I$. Variable $c[i]$ represents the set of messages in the communication channel connecting process p_{i-1} to process p_i that have been sent by p_{i-1} but have not been received by p_i yet.

Any execution of the distributed algorithm is a finite or infinite sequence of states $s_0, s_1, \ldots, s_z \ldots$, such that each pair (s_j, s_{j+1}) is obtained as the consequence of executing a computation step of the algorithm at some process. We assume that in the initial state s_0 of every execution there exists a non-empty subset of processes indexes C such that $\forall i \in C : s_0.status_i = candidate$, whereas $\forall j \in I \setminus C : s_0.status_j = relaying$. Moreover, we write $s_i \rightarrow s_j$ to indicate that s_i happens before s_j in the sequence of reachable states.

A process p_i can execute one of the following computation steps that results in the creation of a new state: either $init_i$ (the initialization described in Figure 1) or one of the possible combinations of code lines 01 and xx (where $xx \in \{$05,06,07,09,10,12$\}$). These computation steps are assumed to be atomic and weak-fair, i.e., if one of them is always enabled to be executed, it is eventually executed.

The following lemma specifies some invariant properties of the reachable states of an execution that will be useful when proving the correctness of the algorithm.

Lemma 1. *For every reachable state s of any execution, and any process indexes $i, j \in I$, the following properties hold:*

(1) $(r, id_i) \in s.c[j] \Rightarrow \forall k \in D(i, j) : s.status_k = relaying$,

(2) $(r, id_i) \in s.c[j] \Rightarrow s.r_i \geq r$,

(3) $(r, id_i) \in s.c[j]$ *and* $(r, id_i) \in s.c[k] \Rightarrow k = j$.

Item (1) of Lemma 1 states that when a message (r, id_i) reaches a process p_j, all processes between p_i (the process that created the message) and p_j must be in the $relaying$ status, excluding both p_i and p_j. Its proof is straightforward, since a message with an identifier id_i can be forwarded by a process with identifier id_k (with $id_i \neq id_k$) only if either p_k was already in the $relaying$ status or it changes to $relaying$ before forwarding (r, id_i). Bear in mind that when a process is in the $relaying$ status it remains forever with this status.

On the other hand, Item (2) of Lemma 1 reflects that if there exists a message (r, id_i), the current round number of the process that created that message, r_i, cannot be lower than r. This is also evident, as when a process creates a message it includes its current round number, and round numbers are never decreased.

Finally, Item (3) of Lemma 1 specifies that each message is unique within the whole ring, i.e., the same message cannot exist in two different communication channels. This is because (a) each message (r, id) is created only once, since process identifiers are all different and when a process creates a new message, it does so with a higher round number than that of the last message it sent; and (b) each message is handled only once by each process.

In order to prove that the proposed algorithm is correct, we first show in Theorem 1 that there cannot be more than one process elected as leader in the execution of the algorithm, i.e., the leader is unique.

Theorem 1. *For every reachable state s of any execution:* $\forall i, j \in I : (s.status_i = leader \wedge s.status_j = leader) \Rightarrow i = j$.

Proof By contradiction, let us consider that $s.status_i = s.status_j = leader$ with $i \neq j$. Assuming that i is the first process that changed to status $leader$, then there exists a previous state s' such that $s'.status_i = candidate$ and the message $(s'.r_i, id_i) \in s'.c[i]$ (note that $(s'.r_i, id_i)$ is the message that has already traveled around the ring and changes the status of p_i from $candidate$ to $leader$ when p_i receives it, with $s'.r_i = s.r_i$). By Lemma 1(1), $\forall k \in D(i, i) : s'.status_k = relaying$. Then, $s'.status_j = relaying$. Since there is no transition from $relaying$ to $leader$ in the algorithm, the theorem holds by contradiction. $\square_{Theorem\ 1}$

Apart from ensuring that the leader is unique, we must prove that the algorithm always terminates electing a leader. First, we show that there cannot exist a state during the computation such that every process is in the $relaying$ status.

Lemma 2. *For every reachable state s of any execution,* $\exists i \in I : s.status_i \neq relaying$.

Proof By contradiction, let us assume that there exists an execution and a reachable state s such that $\forall i \in I : s.status_i = relaying$. Since in the initial state s_0 there is at least one process p_k such that $s_0.status_k = candidate$, state s must happen after s_0. Let p_j be the last process in the execution that changed status from $s'.status_j = candidate$ to $s.status_j = relaying$. At s', there exists a message $m = (s_1.r_i, id_i) \in s'.c[j]$ such that $m \succ (s'.r_j, id_j)$, thus forcing p_j to change to status $relaying$. This message m was created at a state s_1 (previous to s') in which $s_1.status_i = candidate$ and $m \in s_1.c[(i + 1)]$. Between states s_1 and s', process p_i must have changed to the $relaying$ status (recall that p_j is the last process to change to the $relaying$ status).

This argument is now applied to the last state s'_1 in which p_i was in the $candidate$ status. We have that $s_1 \rightarrow s'_1 \rightarrow s'$. At s'_1, there exists a message $m' = (s_2.r_k, id_k)$, with $m' \in s'_1.c[i]$ such that $m' \succ (s'_1.r_i, id_i)$. This message m' was created in a state s_2 (previous to s'_1) at which $s_2.status_k = candidate$ and $m' \in s_2.c[(k+1)]$.

By Lemma 1(2) and the fact that round numbers cannot be decreased, the following relation holds: $(s.r_k, id_k) \succeq (m' = (s_2.r_k, id_k)) \succ (s.r_i, id_i) \succeq (m = (s_1.r_i, id_i)) \succ (s.r_j, id_j)$.

Every process p_k that changes to the $relaying$ status before p_j satisfies this order relation in the left direction of the ring with respect to p_j, i.e., $(s.r_k, id_k) \succ (s.r_j, id_j)$. Therefore, there exists a process, namely p_x, such that it is the first process at the right of p_j such that it was a candidate at some state after the initial state and changed to the $relaying$ status before state s'. Process p_j was a candidate from states s_0 to s'. By the algorithm code, the channels from p_j to p_x can only contain messages originated at p_j. Let $m'' = (s_3.r_j, id_j) \in s_3.c[x]$ be the last message originated at p_j, being s_3 the last state in which $s_3.status_x = candidate$. In order for p_j to change to the $relaying$ status, $m'' \succ (s_3.r_x, id_x)$ must hold. Due to Lemma 1(2) and the fact that $s_0 \rightarrow s_3 \rightarrow s$, it is true that $(s.r_j, id_j) \succeq m'' \succ (s.r_x, id_x)$.

However, recall that we have shown before that for every process p_k that changes to the $relaying$ status before p_j it holds that $(s.r_k, id_k) \succ (s.r_j, id_j)$. Thus, for p_x we have that $(s.r_x, id_x) \succ (s.r_j, id_j)$, which contradicts $(s.r_j, id_j) \succeq m'' \succ (s.r_x, id_x)$. $\square_{Lemma\ 2}$

After showing that not all processes can be in the $relaying$ status, we must prove that the number of candidate processes decreases as the computation evolves until one of them is elected as the leader. This statement is formalized in Lemma 4, which uses Lemma 3 to ensure that there always exists a message with the current maximum round number of the ring.

Lemma 3. *Let s be a reachable state of any execution achieved after executing step $init_i$ for any $i \in I$ such that $s.status_i = candidate$ and $s.r_i = k$ being k the current maximum round number in the ring (i.e., $\forall j \in I, s.r_j \leq k$). Then, there exist $j, l \in I$ such that $(k, id_j) \in s.c[l]$.*

Proof By contradiction, let us assume that $\forall j, l \in I : (k, id_j) \notin s.c[l]$. By the algorithm code, there must exist a reachable state $s_1 \rightarrow s$ at which p_i sent a message containing its round number and identifier; i.e., $s_1.status_i = candidate$, $s_1.r_i = k$, and $(k, id_i) \in s_1.c[i+1]$. The disappearance of message (k, id_i) must have been due to a computation step executed by some process p_j at a reachable state s_2 that results in another state s_3, where $(k, id_i) \in s_2.c[j]$, $(k, id_i) \notin s_3.c[j+1]$, $s_1 \rightarrow s_2 \rightarrow s_3$ and either $s_3 \rightarrow s$ or $s_3 = s$. This is possible if and only if $s_2.status_j = candidate$, $j \neq i$ and $s_2.r_j = k$. Note that: (a) if $j = i$ then $s_3.status_i = leader$ and hence $s.status_i \neq candidate$; (b) if $s_2.r_j > k$ then k is not the maximum round number at s; and (c) if $s_2.r_j < k$ then $(k, id_i) \in s_3.c[j+1]$ and $s_3.status_j = s.status_j = relaying$.

Therefore, if $s_2.r_j = k$ and $id_j(-1)^k > id_i(-1)^k$, then $s_3.r_j = k+1$, thus k cannot be the maximum round number in the ring at s. On the other hand, if $s_2.r_j = k$ and $id_j(-1)^k < id_i(-1)^k$, then $s_3.status = s.status = relaying$.

Let s' be the state previous to s_2 at which p_j sent the message containing its current round number and identifier; i.e., $s'.status_j = candidate$ and $(k, id_j) \in s'.c[j+1]$. By the initial assumption, $(k, id_j) \notin s.c[l]$ for all $l \in I$. By replacing i by j and j by n in the previous argumentation, we conclude that all processes in the ring are in the $relaying$ status, which is a contradiction with Lemma 2. $\square_{Lemma\ 3}$

In the following, $\#candidates(s)$ represents the number of candidate processes in a state s, i.e., $\#candidates(s) = |\{i \in I : s.status_i = candidate\}|$.

Lemma 4. *For every reachable state s of any execution, there exits a reachable state s' such that $s \rightarrow s'$ and $\#candidates(s) > \#candidates(s')$ or $\exists j \in I : s'.status_j = leader$.*

Proof By contradiction, let us assume a reachable state s', $s \rightarrow s'$, where $\forall j \in I : s'.status_j \neq leader$ and $\#candidates(s) \leq \#candidates(s')$.

The algorithm never increments the number of initial candidate processes, i.e., none of the processes p_j such that $s.status_j = relaying$ can become a candidate after s happens. Thus, $\forall s' : s \rightarrow s' : \#candidates(s) = candidates(s')$.

By Lemma 2, $\#candidates(s) > 0$. Being k be the maximum round number at s, by Lemma 3 there exists a message $(k, id_j) \in s.c[l]$ for some $j, l \in I$.

Let p_x be the first process at the right of p_{l-1} such that $s.status_x = candidate$ (note that x may be l). Since (k, id_j) will eventually be received by p_x (which will still be a candidate after processing the message), there exists a first state s_1 such that $s_1.status_x = candidate$, $s_1.r_x = k+1$ and $(k+1, id_x) \in s_1.c[x+1]$, with $s \rightarrow s_1$. Note that if $x = j$, then by Lemma 1(1), $\forall i \in D(x, x) : s.status_i = relaying$. Therefore, either (k, id_x) changes p_x to the $leader$ status or another message (k', id_x) with $k' \geq k$ will eventually change p_x to the $leader$ status after traveling around the ring. These two cases are a contradiction with the initial assumption, thus $x \neq j$.

Let p_y be the first process at the right of p_x such that $s.status_y = candidate$. Processes p_x and p_y cannot be the same, because $(k+1, id_x) \in s_1.c[x+1]$ and $\forall i \in D(x, x) : s.status_i = relaying$, which would make p_x the leader. Hence, we have that $x \neq y$ and $(k+1, id_x) \in s_1.c[x+1]$. This implies that there must be a first state s_2 such that $s_2.status_y = candidate$, $s_2.r_y = k+1$ and $(k+1, id_y) \in s_2.c[y+1]$. This is possible because $s.r_y \leq k$.

In general, every message (r, id_n) with $n \neq x$ is in a channel $c[i]$ with $i \in D(x+1, y)$ satisfying $r < k+1$ and $n \in D(x+1, y)$. These messages were originated at processes at the right of p_x, as otherwise they would be in the $relaying$ status by Lemma 1(1). These messages will be discarded by p_y upon receiving them.

Recall that at s_2, $(k+1, id_y) \in s_2.c[y+1])$. The next candidate process p_n at the right of p_y generates a new message $(k+1, id_n)$ at some state after s. Therefore, there exists a state s_3 with $s_1 \rightarrow s_3$ such that $s_3.status_x = candidate$, $s_3.r_x = k+2$ and $(k+2, id_x) \in s_3.c[x+1]$.

Process p_y will have messages $(k+1, id_x)$ and $(k+2, id_x)$ in its communication channel $c[y]$ at some state s_5, where $s_5.r_y = k+1$. Message $(k+1, id_x)$ will be received first (otherwise p_y would change to the $relaying$ status). Process p_y will only remain as a candidate if $id_x(-1)^{k+1} < id_y(-1)^{k+1}$. Then, upon receiving $(k+2, id_x)$ at state s_6 (where $s_6.r_y = k+2$ because p_y incremented its round number when processing message $(k+1, id_x)$), process p_y will only remain as a candidate if $id_x(-1)^{k+2} < id_y(-1)^{k+2}$, which is a contradiction with $id_x(-1)^{k+1} < id_y(-1)^{k+1}$. $\square_{Lemma\ 4}$

Theorem 2. *For every execution, there exists a reachable state s and a process p_j such that $s.status_j = leader$.*

Proof Every execution starts in an initial state s_0 where at least one process is assumed to be in the $candidate$ status. The execution of successive computation steps will produce new states. As stated in Lemma 4, the number of processes which are in the $candidate$ status decreases as the computation evolves, thus processes change either to the $relaying$ or the $leader$ status until there are no candidates left. According to Theorem 1, if there exists a leader, it must be unique. Moreover, not all candidates can change to the $relaying$ status, as proven in Lemma 2. Consequently, the number of candidates will decrease until eventually only one of them survives. This last candidate will become the leader. $\square_{Theorem\ 2}$

Remark It is important to notice that (differently from Higham-Przytycka's proof) the previous proof of correctness does not assume that the channels are FIFO.

The rest of this section addresses the upper bound issues regarding the number of messages exchanged during the execution of the algorithm depending on whether processes start with the same round number (Theorem 3) or they start with arbitrary round numbers (Theorem 4).

Theorem 3. *Let us consider an execution in which all processes start with the very same round number. The processes exchange at most $O(n \log n)$ messages.*

Proof In order to prove that the number of messages is $O(n\ \log\ n)$ when all the initial round numbers are the same, it is necessary to determine the number of processes needed inside a running cycle of the algorithm so that one of them eventually reaches the greatest round number r_g.

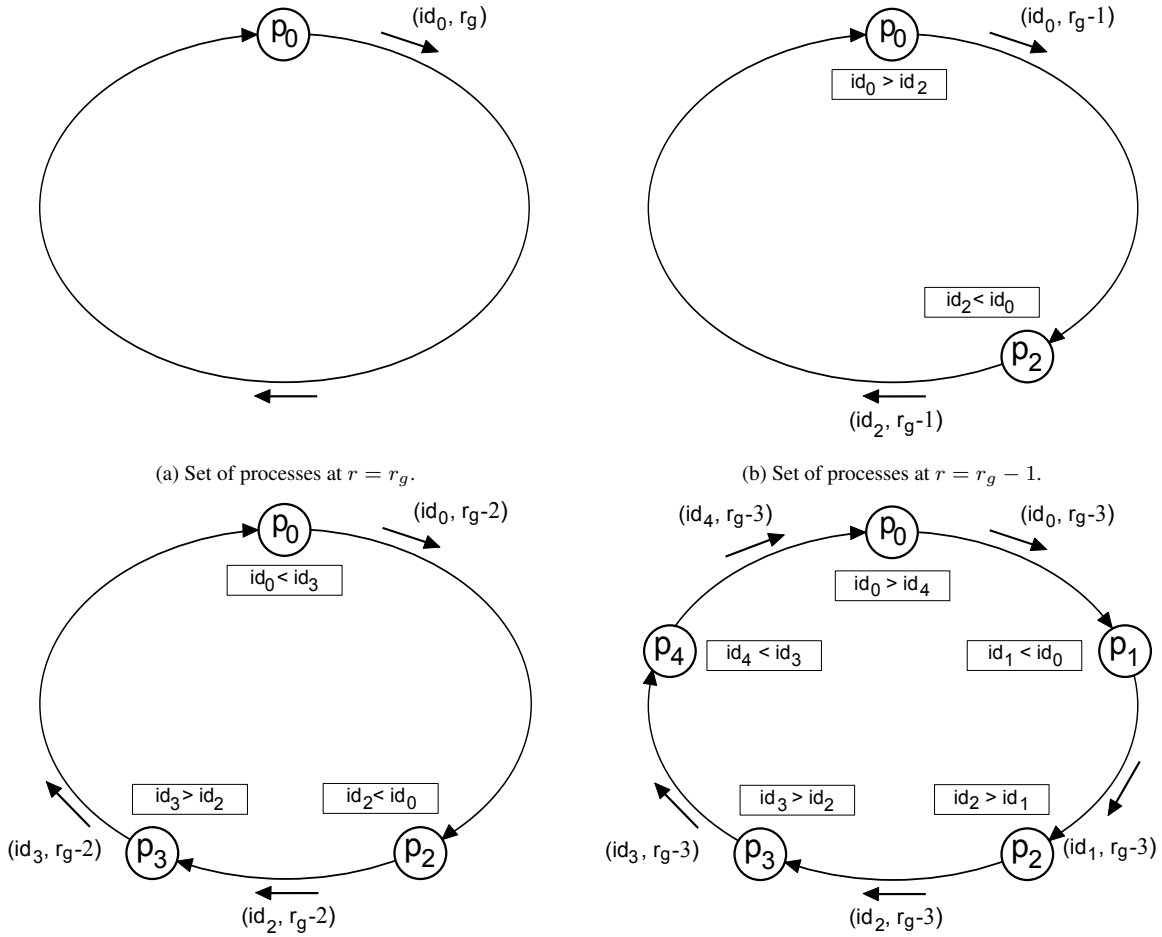
Let us consider a running cycle where a given process p_0 has reached the *leader* status with a round number r_g (see Figure 2(a)). According to the proposed algorithm, there are two cases to consider, depending on whether r_g is odd or even.

Assume that r_g is odd (the same assumptions can be applied to the even case). If p_0 has reached this round number it is due to the fact that there exists another process p_2 with $id_0 > id_2$ ($id_0 < id_2$ for the even case), which sent a message that has been received at p_0, such that $r_0 = r_2 = r_g - 1$, as shown in Figure 2(b). Again, if processes p_0 and p_2 reached round $r_g - 1$, it was because there exists a third process p_3 such that $id_3 > id_0$ and $id_3 > id_2$ (the opposite for the even case). Process p_0 sends a message that is received by p_2, p_2 sends a message that is received by p_3 and p_3 sends a message that is received by p_0, where all their associated round numbers are the same, i.e., $r_0 = r_2 = r_3 = r_g - 2$ (see Figure 2(c)). If these processes have the same round number at that point, there must be at least another two processes p_1 and p_4 satisfying the same conditions as mentioned before. Hence, we have five processes p_0, p_1, p_2, p_3, p_4 such that $id_3 > id_0 > id_4 > id_2 > id_1$, $r_0 = r_1 = r_2 = r_3 = r_4 = r_g - 3$ and the reception of messages is as follows: p_0 receives a message with the same round number as its own but with an identifier id_4 which is lower than id_0; p_4 receives a message with the same round number and an identifier id_3 which is greater than id_4; p_3 receives a message coming from p_2 such that id_3 is greater than id_2; p_2 receives a message from p_1 whose identifier id_1 is lower than id_2; and finally, p_1 receives a message from p_0 whose identifier id_0 is greater than id_1. This new step is depicted in Figure 2(d).

If we continue applying this procedure in an iterative way we will reach a minimum number of processes according to the sequence $1, 2, 3, 5, 8, 13, \ldots$. This succession of numbers corresponds to the *Fibonacci* sequence. Without loss of generality, let us assume that the initial round number is 0. Therefore, if a process p_i has a round number $r_i = p$, then for every integer q, being $q \leq p$, there will exist a set of processes N_{p-q} that will reach a round number $p - q$, given that $\#N_{p-q} \geq fib(q+2)$. More precisely, there exists a set of processes N_0 that started with an initial status $candidate$ and initial round number $r = 0$, such that $\#N_0 \geq fib(p+2)$. This means that if we have x processes whose status at startup time is $candidate$, we can assert that the number of processes that will reach a round number $r = p$ in the worst case is $x \geq fib(p+2)$.

Let us consider a ring with n processes that start at round number 0. Hence, we can delimit the greatest round number L with the value $n \geq fib(L+2)$. By means of the Fibonacci expression:

$$fib(r) = c \cdot (p^r - q^r), \text{ with } p = \tfrac{1+\sqrt{5}}{2},\ q = \tfrac{1-\sqrt{5}}{2},\ c = \tfrac{1}{\sqrt{5}}.$$



(a) Set of processes at $r = r_g$.

(b) Set of processes at $r = r_g - 1$.

(c) Set of processes at $r = r_g - 2$.

(d) Set of processes at $r = r_g - 3$.

Figure 2: Analysis of the communication cost in the worst case of the proposed leader election algorithm

We can set an upper bound for L based on the number of processes initially composing the ring (n) by substituting the aforementioned expression in $n \geq fib(L+2)$:

$$n \geq \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{L+2} - \left(\frac{1-\sqrt{5}}{2} \right)^{L+2} \right).$$

The second term of the difference is upper bounded by the value of $L = 2$, which gives a value of 0.15:

$$\sqrt{5} \cdot n + 0.15 \geq \left(\frac{1+\sqrt{5}}{2} \right)^{L+2}.$$

Applying logarithms to both sides of the inequality results in the following expression:

$$\log(\sqrt{5} \cdot n + 0.15) \geq (L+2) \cdot \log \left(\frac{1+\sqrt{5}}{2} \right).$$

Thus, we can define the upper limit of L for a given set of n processes in the following manner:

$$L \leq \frac{\log(\sqrt{5} \cdot n + 0.15)}{0.21} - 2.$$

This result asserts that the maximum round number that a process can reach is $O(\log\ n)$. Since n messages a re necessary for each round, the total number of messages is $O(n\ \log\ n)$. $\square_{Theorem\ 3}$

Theorem 4. *Let us consider an execution in which all processes start with arbitrary round numbers. The processes exchange at most $O(n^2)$ messages.*

Proof The worst case initialization is when

- We have for $0 \leq i \leq n-2$: $id_i < id_{i+1}$ and the initial values of the round numbers are such that $r_i < r_{i+1}$ (e.g., $id_i = r_i = i$), and
- The orientation of the unidirectional ring is p_{n-1}, p_{n-2}, ..., p_1, p_0, p_{n-1}.

In that case, the algorithm behaves exactly as Chang and Robert's unidirectional leader election algorithm [5] whose message complexity is $O(n^2)$.

More precisely, we have the following. All processes are initially candidates and each process p_i sends the message (i, id_i). The message $(0, id_0)$ sent by p_0 is stopped by p_{n-1} which discards it. The message $(1, id_1)$ sent by p_1 is forwarded by p_0 and then stopped by p_{n-1} which discards it. Etc. until the message $(n-1, id_{n-1})$ sent by p_{n-1} is forwarded by p_{n-2}, p_{n-3}, ..., p_1, p_0 which forwards it to p_{n-1}, which elects itself as a leader. The total number of sent/forwarded messages is consequently $1 + 2 + \cdots + (n-1) + n$, i.e., $O(n^2)$.

Remark. This initialization is the worst because it is the one that favors message forwarding as much as possible. $\square_{Theorem\ 4}$

5 Conclusion

This paper has introduced and investigated the notion of leader election with graceful degradation. In the context of leader election algorithms, graceful degradation is related to the initial values of local variables and message complexity. More precisely, a gracefully degrading leader election algorithm has to behave optimally when the local variables have "good" initial values, while it may not behave optimally (while still electing a leader) when local variables have "bad" initial values. Optimal means $O(n \log n)$ messages, while non-optimal means $O(n^2)$ messages.

The paper has presented such a gracefully degrading leader election algorithm. This algorithm, based on a principle introduced by Higham and Przytycka in [7], is optimal with respect to the number of messages when the processes start with the same round number while it can require up to $O(n^2)$ messages when the processes start with arbitrary round numbers.

Finally, beyond the leader election problem, the study of algorithms whose complexity gracefully degrades according to the initial values of local variables is an approach that seems worth pursuing from both practical and theoretical point of views.

References

[1] Angluin D., Local and Global Properties in Networks of Processors. *Proc. 12th ACM Symposium on Theory of Computation (STOC'81)*, ACM Press, pp. 82-93, 1981.

[2] Attiya H., Snir M. and Warmuth M., Computing on an Anonymous Ring. *Journal of the ACM*, 35(4):845-876, 1988.

[3] Attiya H. and Welch J., Distributed Computing: Fundamentals, Simulations and Advanced Topics (2d Edition). *Wiley-Interscience*, 414 pages, 2004.

[4] Bodlaender H.L., Some Lower Bound Results for Decentralized Extrema Finding in Ring of Processors. *Journal of Computer System Science*, 42:97-118, 1991.

[5] Chang E.J.H. and Roberts R., An Improved Algorithm for Decentralized Extrema Finding in Circular Configurations of processes. *Communications of the ACM*, 22(5):281-283, 1979.

[6] Dolev D., Klawe M. and Rodeh M., An $O(n \log n)$ Unidirectional Algorithm for Extrema Finding in a Circle. *Journal of Algorithms*, 3:245-260, 1982.

[7] Higham L. and Przytycka T., A Simple Efficient Algorithm for Maximum Finding on Rings. *Information Processing Letters*, 58:319-324, 1996.

[8] Hirschberg D.S. and Sinclair J.B., Decentralized Extrema Finding in Circular Configuration of Processors. *Communications of the ACM*, 23:627-28, 1980.

[9] Le Lann G., Distributed Systems: Towards a Formal Approach. *IFIP World Congress*, pp. 155-160, 1977.

[10] Peterson G.L., An $O(n \log n)$ Unidirectional Algorithm for the Circular Extrema Finding. *ACM Transactions on Programming Languages and Systems*, 4(4):758-762, 1982.

[11] Raynal M., Algorithms for Mutual Exclusion. *The MIT Press*, 107 pages, 1986.

[12] Raynal M., Communication and Agreement Abstractions for Fault-Tolerant Distributed Systems. *Morgan & Claypool Publishers*, 265 pages, 2010, ISBN 978-1-60845-293-4.

[13] Santoro N., Design and Analysis of Distributed Algorithms. *Wiley-Interscience*, 509 pages, 2007.